ChatGPT

Implementation Strategy for a Smart Tourist Safety System

Overview and Objectives

This solution will integrate **blockchain-based digital IDs, geo-fencing, AI anomaly detection, and IoT wearables** into a cohesive platform focused on tourist safety. Tourists will register and receive a secure digital ID, then use a mobile app that monitors their location (with consent) and alerts them and authorities of any danger. Authorities (tourism and police) will have a live dashboard to track tourists (in aggregate) and respond swiftly to incidents. The primary goal is to **ensure tourist security in real-time** – especially in remote or high-risk areas – while respecting privacy and providing a user-friendly experience. This is particularly crucial in regions like Northeast India, where difficult terrain and limited connectivity make traditional monitoring challenging [1]. The system is designed for tourists as the main users (via the mobile app and optional wearables), with police and tourism officials as secondary users (dashboard and response tools). Below, we outline the architecture, tech stack, MVP plan, and detailed requirements.

System Architecture and Components

1. Blockchain Digital ID Platform: Tourists enroll at entry points (airports, hotels, etc.) by providing KYC information (e.g. passport or Aadhaar details) and trip specifics. A **digital tourist ID** is issued on a blockchain for tamper-proof record-keeping [2]. Each ID could be represented as a unique token or entry on the ledger, containing a reference to the tourist's verified identity, trip duration, and emergency contacts. The blockchain (likely permissioned, managed by authorities) ensures the ID data cannot be altered without detection, enhancing trust in the system [3]. Tourists might receive a QR code or digital certificate to use as identification, which can be scanned by authorities for verification. Using a decentralized identity approach, tourists remain in control of their personal data sharing – only authorized officials can retrieve the full details, and only during the valid period of the visit [4]. (After the visit, the ID expires or is revoked to protect privacy.) This component will leverage the immutability and security of blockchain to prevent identity fraud and provide an auditable trail of tourist registrations [5].

2. Tourist Mobile Application: A smartphone app (initially Android, with plans for iOS) will be the tourist's safety companion. Key features include:

- **Tourist Safety Score:** The app automatically computes a "safety score" for the tourist based on real-time factors like their location history, time of day, and the safety rating of areas visited. For example, visiting well-populated, low-crime areas might keep the score high, whereas venturing into remote or "high-risk" zones late at night would lower the score. (High-risk zones are defined by authorities – e.g. politically sensitive areas, known disaster-prone sites, or border regions – and stored in the system.) This gives travelers immediate feedback; a low score could trigger advice or warnings in the app. The score calculation can start simple (rule-based) and later incorporate AI patterns from past tourist data. Initially, we will implement it as a weighted formula – e.g. **Base 100**

minus points for risk factors such as time in restricted areas, sudden itinerary changes, lack of communication, etc. – to keep it transparent and tunable [6].

- **Geo-fencing Alerts:** Using the phone's GPS, the app monitors if the tourist enters any designated **unsafe or restricted area**. If they cross a virtual boundary of a geo-fenced zone, the app will immediately alert the tourist ("You have entered a restricted/high-risk area – exercise caution or turn back") and simultaneously notify the central system or nearest police unit [6]. For example, if a tourist wanders near an international border or a known insurgency-affected locality, an alert is generated. This uses platform geo-fencing APIs (e.g. Android's Geofence API or Google Maps services) to continuously check location against a list of coordinates for no-go zones. The geofence definitions will be maintained by authorities in the backend (police can update high-risk zones as situations change). Early on, we will hard-code a few sample geofence regions for the MVP demo. As a proof of concept, **entering one of these zones will trigger a push notification on the phone and log an alert on the authority dashboard**. Geo-fencing technology is widely used for safety; for instance, smartphone apps can automatically identify when a user goes out of a safe boundary and send alerts [7]. This feature will be critical in preventing tourists from unknowingly entering dangerous areas.

- **SOS Panic Button:** The app will have an obvious **panic button** (SOS) that a tourist can press in an emergency. Activating it will instantly send the tourist's live **GPS location and profile details to the nearest police control room and their emergency contacts**, while also sounding an alarm on the phone for immediate local help [6]. This is similar to India's **112 Emergency app**, which on SOS activation automatically transmits the user's identifying info and location to authorities for rapid response [8]. In our system, when SOS is triggered, the backend will create a high-priority incident entry viewable on the police dashboard (flashing indicator on the tourist's icon or a separate incident list) and possibly send an SMS/call to local police units if internet connectivity is unreliable. The app can also dial a helpline number (like 112) in parallel as a fallback. For the MVP, we will implement the SOS as a button that, when pressed, sends an **HTTP request to the server with the user's ID and location**, and then displays to the user that help is notified. This simple implementation will demonstrate the concept; integration with actual emergency dispatch systems can follow.

- **Real-Time Tracking (Opt-in):** For tourists who consent (especially useful for solo travelers or by request of family), the app can continuously share their location with selected people or authorities. If enabled, the backend will receive periodic location pings (e.g. every few minutes) and update the tourist's trail on the dashboard map. Families could have a separate interface or code to monitor their loved one's journey in real time (possibly via a secure web link or within the app). This feature will be off by default (to respect privacy), and clearly permission-based. We'll include an **"Allow live tracking?" toggle** in the app – if on, location updates are sent to the server more frequently and stored. For MVP simplicity, we might default to on for demonstration, but in a real product this would be a user setting at registration.

- **Other App Features:** The mobile app will support **multilingual UI** from day one. Given India's diversity, we plan the app strings in English and a few Indian languages (like Hindi, Assamese, Bengali, etc.) initially, and allow easy switching. This can be done by using language files or Android string resources for each locale. We'll also include **voice/text accessibility** – e.g. the SOS feature could be activated by a voice command ("Help") or by pressing power button 3 times (mirroring existing emergency apps) for users who can't navigate the app in a crisis [9]. These are stretch goals

for MVP – at minimum we will design with localization in mind and perhaps implement one alternate language to show the concept.

3. AI-Based Anomaly Detection: A backend service will continuously analyze incoming data (location trails, behavioral patterns) to flag **anomalies that suggest a tourist may be in trouble** [10] . Early warning signs include:

- **Sudden drop-off in location** – e.g. if a tourist's phone shows a rapid descent or fall (detected via GPS altitude or phone sensors) followed by no movement. This could indicate an accident (fall from a height or into water, etc.). The system can detect this by looking at GPS coordinates and elevation: if a person's altitude drops sharply or their location signal disappears abruptly (no signal for X minutes after a high movement speed), that's flagged as a potential incident.

- **Prolonged Inactivity** – if we stop receiving any location or phone activity from a tourist for a prolonged period beyond a threshold (for instance, 6-8 hours of no check-ins or movement during daytime without any known downtime in itinerary). Especially if they were in a non-hotel location before going silent, this could indicate a lost or incapacitated person. The AI module will maintain a heartbeat – expecting periodic GPS or network signals. If a device hasn't reported in and the tourist hasn't reached a known safe location, it generates an alert. (The app can also proactively warn the user, e.g. "We haven't heard from you in a while, do you need help?" to which if no response, escalates the alert.)

- **Deviation from Planned Route/Itinerary** – during registration, the tourist's intended travel plan is recorded (places and dates). If the person significantly strays from this plan (for example, they travel to a different city or attraction not on their itinerary, or do not arrive at a booked destination on the expected date), the system will note it. Not every deviation is an emergency, but combined with other signals (like entering a risky area or going silent), it could be cause for concern. The AI can cross-reference the current GPS with the itinerary. For MVP, we will implement a simple logic: if the tourist is >50km away from any planned itinerary location (and that area is known to be unsafe or off-route), flag it. In future, a machine learning model could be trained on typical tourist movement patterns vs. abnormal ones to improve accuracy.

- **Distress Behavior** – e.g. erratic movement (perhaps running back and forth), or the phone being switched off immediately after being in a high-risk zone could be suspicious. We might not fully implement this in MVP due to complexity, but conceptually an ML model could learn from past incidents to detect "unusual" movement sequences. For now, rule-based triggers as above will suffice (which essentially cover missing or silent behavior) [10] .

When the AI module (or simple rules engine) flags an anomaly, it will create an **alert for authorities** similar to an SOS (but system-generated). For example, "Tourist ID 1234 hasn't moved for 12 hours since visiting XYZ forest." These alerts appear on the dashboard and can be sent as SMS/email to designated officers automatically. The system might attempt to contact the tourist via the app (a push notification like "Are you OK?") – if they confirm all is well, the alert can be resolved; if not, it escalates. By leveraging AI for anomaly detection, the system can **proactively find tourists in distress even if they cannot send an SOS themselves**, a key advantage in remote safety [11] .

4. Tourism Dept & Police Dashboard: Authorized officials will use a secure web-based dashboard to monitor and manage incidents in real time [12]. The dashboard will have several modules:

- **Live Map of Tourists:** A map (likely built with Google Maps or an open-source alternative like Leaflet) shows the real-time or last known locations of all active tourists in the region. Tourists could be represented by dots or icons, possibly color-coded by their safety score or alert status. The map can aggregate tourists into clusters when zoomed out (for an overview of tourist density) and show individual positions when zoomed in. This provides situational awareness of how many visitors are in various areas. We will also overlay **heatmaps for high-risk zones** and possibly weather or hazard information. If an incident occurs (SOS or AI alert), that tourist's icon can flash or change color to draw attention.

- **Alert/Incident Panel:** A list of all current alerts (SOS calls, anomaly detections, geo-fence breaches) with timestamps and details. Each alert entry can be clicked to focus the map on that tourist and show their profile. From here, authorities can acknowledge the alert, assign it to a responder, or mark it resolved. The alert record includes the tourist's ID info (from blockchain record: name, photo if available, emergency contact, etc.), their itinerary, and last known coordinates [12]. For missing person scenarios, the system can automatically compile an **electronic FIR (First Information Report)** for a missing person case – essentially a pre-filled report with the tourist's details, last location, and circumstances, which can be sent to the police database or printed [12]. (FIR is a formal police case registration in India; automating this saves time in emergencies.) The MVP will simulate E-FIR generation by producing a PDF report or a summary on screen that an officer would otherwise have to write manually.

- **Tourist Database Access:** The dashboard lets officials search or click on any tourist's digital ID record. This pulls up the **digital ID details from the blockchain** (via a query interface) or from a mirrored database. Officials can verify a tourist's identity and view their provided documents (like passport scan, if stored off-chain) instantly, which is useful if a tourist needs assistance or is found without physical ID. The blockchain ensures these records are **tamper-proof and trustworthy**, as every issuance and update is logged permanently [13]. Only authorized logins can retrieve personal info, ensuring privacy. For faster access in the MVP, we might store a cache of ID details in a secure database that syncs with the blockchain records (since reading from blockchain might be slower). Viewing an ID might also show that person's *history of alerts* (if any) and current safety score.

- **Administrative Functions:** Authorities can use the platform to update high-risk zone definitions (drawing a geofence on the map and marking it "restricted"), send broadcast messages to tourists (for example, if a sudden landslide or curfew happens, they can push a notification to all tourists in a certain area), and manage user roles. These features ensure the system is not static but can adapt to real-world changes and allow coordination.

Security for the dashboard is paramount – each officer will have login credentials and role-based access (tourism officials may have read-only access to data analytics, while police have incident response capabilities, etc.). All data views and actions will be logged for audit trail.

5. IoT Wearable Integration (Optional for MVP): In high-risk environments (treks, caves, remote trails), not all tourists will have their phones handy or usable (battery could die, or in some cases phone signals are weak). **Wearable devices (smart bands or tags)** can provide an extra layer of safety [14]. We propose a

simple IoT device that a tourist can wear – e.g. a wristband with an SOS button and basic sensors. This device would pair with the tourist's phone via Bluetooth **or** communicate over Wi-Fi/LoRaWAN to the server. For MVP, given we have two ESP32 Wi-Fi modules available, we will prototype a **smart band** using an ESP32. The ESP32 can be programmed to: read a sensor (for instance, a accelerometer or heart-rate sensor) and send periodic data, and listen for a button press to send an SOS. In practice, if a tourist is in a dense forest with no cell service, these bands might connect via long-range radio to gateways – but in our prototype, we will assume the band can connect via the tourist's phone (BLE) or any available Wi-Fi.

Functionality of the IoT integration: - The **band continuously streams health and motion data**. For example, it could send heart rate and movement status every minute. Sudden vital sign anomalies (like a heart rate dropping too low or spiking extremely high) or a detected fall (from an accelerometer) can trigger an automatic distress signal [15]. This could alert the system even if the tourist is unable to press SOS (e.g. they faint or are injured). Such devices exist – many smartwatches today have fall detection that auto-calls help [16]. Our band will demonstrate this concept: if a simulated vital sign crosses a threshold, the ESP32 will call the same alert API as the app would.

- The band has a **physical SOS button** as well. Pressing it will cause the ESP32 to send an immediate alert (via the phone or directly over internet if Wi-Fi-connected) with the tourist's ID. In a real deployment, the band might include a GPS chip to send location too. If not, the system can use the last known phone GPS or a nearby gateway's location. The MVP ESP32 will likely be coded to hit an HTTP endpoint on our server when the button is pressed, including a unique device ID. The backend will map that device to the tourist (this mapping set up during ID issuance – e.g. tourist ID 123 is given device MAC XYZ, stored in their profile). The alert then is treated just like an app SOS.

- **Hardware details:** Using the ESP32, which has Wi-Fi, we will implement a small firmware (in Arduino C++ or MicroPython) that connects to a hotspot (the tourist's phone or a portable MiFi device). When a certain GPIO button is pressed, it makes an HTTPS request to the cloud endpoint. We can also use MQTT (a lightweight IoT messaging protocol) – run an MQTT broker on the server, have the device publish an "SOS" topic message, which the backend subscribes to and processes. MQTT is suitable for IoT due to low overhead. If time permits, we will try out MQTT for demonstration. Otherwise, a simple REST call will suffice.

This IoT integration, while optional, is **important to include in the MVP to showcase a complete solution** as requested. It will be kept workable and simple. With two ESP32s, for example, we could demonstrate one as a **wearable band** transmitting vitals and SOS, and maybe use the second to simulate a **gateway** or just a second user's device. The IoT devices underscore the extensibility of the system – in the future, tourists could even rent these safety bands at park entrances, and they'd work even if phones are off (some bands could use satellite or LoRa networks for off-grid communication). For now, the ESP32 devices will prove that we can integrate external signals into the platform easily [14].

Technology Stack Recommendations

Choosing the right tech stack is critical to meet the requirements of security, real-time operation, and scalability. Based on the problem statement and target environment, here are our suggestions:

- **Blockchain Platform:** We recommend using a **permissioned blockchain** framework like **Hyperledger Fabric** for the digital ID system. Fabric allows defining a private network (members

could include the Ministry of Tourism, Ministry of Home, and state police nodes) ensuring only authorized participants validate transactions. It supports smart contracts ("chaincode") which we can use to implement the ID issuance and verification logic. Each tourist ID could be an entry on the ledger with necessary fields, and the immutability of Fabric ensures tamper-proof records [5]. Fabric also offers good performance (throughput) and fine-grained access control (channels, private data collections) which is useful for handling personal data securely. Alternatively, **Hyperledger Indy** or **Ethereum (Quorum)** could be considered: Indy is built specifically for decentralized identity (DID) management with features for verifiable credentials, which aligns well with issuing a digital ID token that the tourist holds. Quorum (an enterprise Ethereum) would allow smart contracts in Solidity for ID, but being permissioned and with privacy features. For a quick MVP, we might not set up a full multi-node blockchain; instead, we could simulate with a local Ethereum test network or even a simple database with hashing for concept. However, the final recommendation is to use Fabric or Indy because government-grade ID needs trust and privacy (Fabric is already used in some Indian government projects). **Blockchain integration in MVP:** We'll likely start with **Ethereum testnet** using a smart contract for simplicity (since developer tools and documentation are readily available). The contract will store a mapping of tourist ID to a hash of their info. When a tourist registers, we add an entry (transaction) and get back a transaction ID or token ID. Verification involves reading that data. This demonstrates the concept of immutability. In parallel, we will prepare the design for a Hyperledger solution post-MVP. The blockchain layer will interface with the rest of the system through an API service that the backend can call (to issue ID or query ID).

- **Backend Server & AI Logic:** For the server side that ties everything together (receives app data, runs anomaly detection, serves the dashboard API, etc.), a **Node.js/TypeScript** or **Python** stack would be suitable. Node.js with Express can easily handle REST APIs for the app and dashboard, and also manage WebSocket connections if we need real-time push updates (e.g. to live-update the dashboard map as locations stream in). It has good libraries for interfacing with Ethereum (web3.js) or Hyperledger, and for database connectivity. On the other hand, Python (with Flask/FastAPI or Django) is excellent for AI/ML integration. We could even use Python microservices for the AI anomaly detection specifically, if using libraries like scikit-learn or TensorFlow for any predictive model. A possible architecture is to have **Node.js as the main API server** (serving mobile and web requests, handling blockchain transactions via an SDK, and orchestrating events), and a **Python service for anomaly detection** that subscribes to location updates (perhaps via a message queue) and raises alerts. Given the time constraints, we might start with a single server doing both, using Python for rapid development. Python's advantage is quick scripting of anomaly logic (e.g. we can quickly implement a schedule that checks each tourist's last update and flags issues). There are also existing algorithms for anomaly detection we might leverage in the future (like an isolation forest for outlier detection in movement data), but initially a custom rule set is fine. **Database:** We will need a database to store operational data – we suggest using a **PostgreSQL relational database** (or MySQL/MariaDB) for structured data like user profiles, itineraries, and alerts. SQL is good for ensuring consistency (e.g. linking tourist IDs to their emergency contacts and IoT device IDs, etc.). We may also use a **NoSQL store (MongoDB)** for logging frequent location updates or for flexible data (like storing the history of movements as a JSON). In MVP, we might get away with just one type of DB. PostgreSQL with PostGIS extension could be very powerful as it can store geolocation points and perform geo-queries (helpful for checking if a point is within a geofence polygon). PostGIS could, for example, let us run a query "find all tourists inside forbidden zone X" easily. This is a nice-to-have optimization; MVP can simply compute geofence crossings in code.

- **Mobile App Development:** We aim for cross-platform support due to diverse user base, so **Flutter** is an excellent choice – one codebase for Android and iOS, and it has good packages for geolocation, background services, and multilingual support. Flutter's widgets also make it easier to implement clean UI in multiple languages. Additionally, Flutter can compile to web if needed (for a tourist web portal). Alternatively, since Android will be the majority (especially in developing regions) and for faster native access to sensors, a **native Android app (Java/Kotlin)** is a straightforward path for MVP. We can leverage Android's native APIs for location and geofencing. We will also need background location permission (so the app can send updates even if not open, especially for the tracking feature and anomaly detection). For the SOS button, the app will require permission to send SMS or call (for 112 integration), which we can request. In terms of map integration in the app (to possibly show the tourist their own path or nearby help centers), we can use Google Maps SDK. However, maps on the tourist side are secondary (the tourist generally just needs warnings and the panic button – they don't need to see all data). Thus, the app UI will be relatively simple: a home screen showing their safety score and maybe tips ("You're in a safe area. Nearest help: XYZ"), an Alerts screen (any messages from authorities), and the SOS button prominently. Also a profile screen for their digital ID QR code and trip info. Flutter can handle all these. For multi-language, Flutter's localization features or Android's string resources will be utilized. We will gather translations for key phrases (we can get help from online sources for major languages).

- **Web Dashboard:** For the authority dashboard, a modern web framework like **React** or **Angular** will work. We can create a single-page application that calls backend APIs for data and uses WebSockets for real-time updates. **React** with libraries like Leaflet or Google Maps API for React will let us embed the map and markers. We'll also use chart libraries (maybe Chart.js or D3) for any stats (if we show tourist counts over time, etc.). The web app will be behind a secure login. Since this is an internal tool, we can host it on a secure government server or cloud with HTTPS. The tech choice here can be based on team expertise; React is common and would suffice for MVP. We'll ensure the UI is intuitive – e.g. using a sidebar for menu (Map, List of Incidents, Search Tourist, Settings) and the main pane for the map or list views. Performance considerations: the map should handle a few thousand points (in peak tourist season there could be that many tourists). Clustering will help. Also, updates should not overload – we might batch location updates (e.g. the dashboard refreshes positions every 30 seconds or on demand rather than truly every second, to reduce load).

- **IoT and Hardware:** The ESP32s will be programmed using **Arduino IDE (C++)** for simplicity. We'll use libraries for Wi-Fi connectivity and maybe an HTTP client or MQTT client. If needed, we might connect a **sensor** like a simple heartbeat sensor or use the built-in analog input to simulate one (we can just vary a potentiometer to simulate heartbeat changes). The hardware stack might involve: ESP32 -> (via Wi-Fi) -> API endpoint (or MQTT broker on the backend). If using MQTT, we can embed an MQTT broker like Mosquitto on the server or use a cloud IoT hub (for MVP, local Mosquitto is fine). The advantage is that the ESP32 can publish data and the backend can get it in real-time with low overhead. We'll have to ensure encryption even in IoT communications – e.g. use TLS for MQTT or HTTPS for REST, to prevent spoofing. Since the devices are prototypes, we will at least simulate secure tokens or basic auth in their requests. In the future, each device could have its own crypto key pair and auth method (which could even be tied to the blockchain ID, e.g. sign messages to prove authenticity). Those are advanced measures beyond MVP scope.

- **Mapping and Location Services:** Google Maps and its related APIs (Places, Directions) could be useful. For instance, if we want to incorporate route safety, Google's Directions API could suggest

safer routes or at least give route info (though "safety" is not a standard metric there). We might also use **OpenStreetMap data + custom layers** for marking safe vs unsafe zones. If internet is an issue in some areas, the app could have offline map tiles preloaded for key regions (ensuring that even if data is offline, the geofence logic still works locally). For MVP, we will assume connectivity for simplicity, but design mindful that the app should handle intermittent connectivity (cache last known safe zones, queue alerts to send when back online, etc.).

- **AI/ML Tools:** While initial anomaly detection is rule-based, we plan to enhance it with AI. For example, training a model on past tourist trajectories that ended in incidents vs those that didn't, to predict risk. We might use Python's **scikit-learn** for anomaly algorithms or even frameworks like **TensorFlow/PyTorch** for sequence models (e.g. an LSTM that watches a sequence of location coordinates and outputs a risk score). These require data that we won't have initially, so the plan is to accumulate data and then refine the AI. Meanwhile, simpler techniques like threshold-based alerts and maybe an **Isolation Forest** (an unsupervised anomaly detector that could run on features like "distance from itinerary", "time of no movement") could be experimented with. The tech stack easily accommodates this: a Python service can run these periodically. We will also incorporate some **GIS analytics** – e.g. if multiple tourists trigger SOS in the same area, the system could recognize a pattern (maybe a local hazard) and notify all others to avoid that area. These are future capabilities.

- **Security & Data Privacy:** We will enforce end-to-end encryption for data in transit. The app will communicate over HTTPS to the backend. User data such as passport details will be encrypted at rest (e.g. in the database) and only decrypted when needed by an authorized request. Since blockchain data is public in some cases (if Ethereum public network were used), we will **never put raw personal info on-chain** – instead we put a hash or an index, and retrieve actual data from a secure off-chain store. A **Zero Trust architecture** will be followed: even internal components authenticate and authorize. Tourists will log in to the app possibly via their mobile number/OTP or using their digital ID (which might be a DID and key). The authorities' dashboard will have multi-factor authentication given the sensitive info accessible. Also, the entire design will comply with relevant data protection laws (for example, India's Personal Data Protection provisions): data collection is minimal (only what's needed for safety), and data is retained only for the duration necessary (tourist data might be deleted or archived once their trip is over, unless an incident requires keeping it). Blockchain's immutability might conflict with deletion, so the on-chain part will only contain non-personal pointers, enabling us to delete the off-chain personal details when no longer needed, thus maintaining privacy compliance [17] [18].

In summary, the stack is **Hyperledger Fabric (Blockchain)**, **Flutter mobile app**, **Node/Python backend with PostgreSQL**, **React web dashboard**, and **ESP32 IoT devices** – leveraging APIs like Google Maps for geo-services and focusing on secure, scalable design. This combination balances innovation (AI, blockchain, IoT) with practical development (leveraging existing frameworks and cloud services where possible).

MVP Development Roadmap

To build a **Minimum Viable Product (MVP)** that demonstrates the core functionality, we will follow a phased approach. Since this is a proof-of-concept, we focus on the most crucial features first, ensuring each component works end-to-end, then iterate to add more. Below is a suggested MVP roadmap:

Phase 1: Core Functionality (Digital ID, Basic App, Basic Dashboard)

- **Digital ID Issuance (MVP Simplification):** Implement a basic registration web form or admin tool where we input a tourist's details (name, ID proof number, etc.) and itinerary. Instead of a full blockchain integration on day one, we might start by generating a *mock digital ID* (like a UUID or QR code) and storing the info in our database. At the same time, we'll prepare a smart contract on an Ethereum testnet that can record a hash of this data – and execute a transaction to simulate the blockchain record. For the demo, we'll show that the blockchain has an entry (e.g., via a block explorer or a query) corresponding to the issued ID [3]. This proves tamper-proof logging. In Phase 1, this can be done for one or two sample tourists manually. The tourist's digital ID (perhaps a QR code encoding their unique ID or blockchain transaction ID) will be output by the system. We will ensure the ID has an expiration date attribute.

- **Mobile App Basic Version:** Develop the app to allow a tourist to "log in" with their issued digital ID or a simple credential. Core features to implement first: **live location tracking** (the app obtains GPS coordinates periodically and sends to server), and the **SOS button**. Geofencing can be initially tested by hardcoding one geofence area – for example, we designate a small area near us as "restricted" and if the user walks into it, the app triggers an alert to the server. We will also implement display of the user's ID info (maybe a screen that shows their name, photo, ID expiry – effectively a digital ID card on screen). The safety score calculation can be rudimentary in this phase (e.g., a fixed value or simple logic since we won't have enough movement data yet). The **SOS workflow** will be fully implemented: when pressed, it sends location to server and the app shows a "Help is on the way" message. We'll simulate emergency contact notification by perhaps having the server send an email or SMS (using a third-party service like Twilio or simply logging it) to a predefined contact. The app UI in Phase 1 will be minimal but functional – a map view isn't strictly needed yet, but maybe a status text ("All Safe" vs "Help sent!" etc.).

- **Backend and Alert Handling:** Set up the backend server to receive data from the app. This includes endpoints: `/locationUpdate`, `/panicAlert`, `/geofenceBreach` etc. For MVP, a single endpoint might suffice that handles any alert type. When an alert comes in, the server stores it in the database and also pushes it to any listening dashboard clients (websocket). The anomaly detection at this stage will be manual or very simple (we may skip it in Phase 1 and add in Phase 2). The backend should also expose an API for the dashboard to fetch tourist data and current locations. We will verify that the app successfully sends data and the backend correctly logs it.

- **Dashboard Basic Version:** Create a simple web page that can display at least: a list of active tourists (for MVP could just be a table with name, ID, current location coords, status) and a list of alerts. We will focus on the **incident/alert view** first. For example, whenever the SOS is pressed, the new alert should appear on the dashboard (without needing a refresh). We'll implement this with a WebSocket or by periodic polling in the web app. The dashboard in Phase 1 can be very basic in design – even a simple HTML/JavaScript page is fine – because the goal is to prove that an alert generated on the app is immediately visible to an authority. We'll also add a button on the dashboard to simulate **E-FIR generation**, which for now could just pop up the details of the missing person (tourist profile and last known location) that an officer can use.

- **Testing Phase 1:** We will run a scenario with one test tourist: Register them -> open app -> simulate normal movement (send a couple of locations) -> trigger an SOS -> see it on dashboard. Also simulate going into a restricted zone if possible (we can fake this by sending a location that we know is "inside" our test geofence). Ensure alerts show up. This completes the core safety loop.

Phase 2: Enhanced Features (Geo-fencing, Anomaly AI, IoT device)
- **Geofencing Full Implementation:** Expand the geofence logic to support multiple zones and integrate it properly. In Phase 2, we'll have the backend store geofence coordinates (perhaps as polygons). The mobile app can either download these and do local checks (more efficient) or send its location to backend and let the backend check (simpler initial approach). We likely do local checks using a geofencing API – meaning the app registers geofences with the OS, which then wakes up the app if a boundary is crossed. We'll test this by marking a small radius around our location as "restricted" and physically walking outside/inside it to trigger. The alert flow for geofence entry will be similar to SOS (an alert but of type "Geofence" automatically). Additionally, implement an **on-exit geofence** event if needed (for example, to know when they left a safe zone).

- **Tourist Safety Score Logic:** Now that location tracking is running, we can start updating a "safety score" on the fly. Develop a simple algorithm: e.g., start each day at 100; if tourist enters a high-risk zone, subtract 30; if out after midnight in unfamiliar area, subtract 20; if stays on planned route, add 10; etc. We'll display this score on the app and also on the dashboard (maybe color-coded: green for 80-100, yellow 50-79, red below 50). This score is more of a guide – in the MVP we'll manually manipulate it to demonstrate (like triggering conditions that lower it).

- **AI Anomaly Detection Service:** Build out the anomaly detection module. For MVP demonstration, we can create a scheduled job (running every X minutes) that checks each tourist's last update time and location vs itinerary. If last update > e.g. 1 hour (configurable) and tourist hasn't indicated stopping (like reaching hotel), flag an alert "No contact from [Name] for 1 hour". We can also implement a simple route deviation check: pick a tolerance (say 30 km). If the tourist's last known location is 30+ km away from any expected location for that day, create an alert. We'll demonstrate this by intentionally deviating our test tourist's location from their plan. This module can be a Python script integrated with the database. In future, this would be replaced or augmented by machine learning. But even rule-based checks add a lot of value (many safety issues can be caught by these patterns).

- **IoT Wearable Integration:** Now incorporate the ESP32 device. We will code the ESP32 to periodically send a signal (like heart rate = 72 bpm) to an `/iot` endpoint or via MQTT topic. This shows up on the system – we might display in the dashboard the latest "health" of that tourist (or simply log it). More importantly, wire up the SOS button on the device: pressing it triggers the same alert flow. We will do a live demo where a team member presses the hardware button and we show on the dashboard that an SOS was received from the IoT band (with the tourist identity). This proves that even without using the phone app, an alert can go out. We'll have to ensure the device knows which tourist it belongs to – likely by hardcoding for demo, or we scan the device's ID at registration time. Because the user specifically has two ESP32, we can have Tourist A with a wearable that sends data, and Tourist B as just phone user, to show two concurrent users in the system.

- **Dashboard Enhancements:** Improve the dashboard UI now – integrate a map view using a mapping library. Plot the tourist locations with markers. Implement basic clustering or at least ensure performance with a handful of points. Add the ability to click on a tourist marker to see their details (pop-up with name, ID, etc.). Also, build a filter to show only active alerts or select a tourist from a list. We'll also add a **heatmap overlay for high-risk zones** for visualization (this can be done using semi-transparent red polygons on the map for each restricted area). If possible, incorporate a toggle to show "safety scores": e.g., a view where all tourists with low scores are highlighted, to help officials

proactively check on them. Another feature to implement: an option to **resolve or acknowledge an alert** on the dashboard. For MVP, clicking "resolve" could just remove it from the active list (and maybe log the time). In a real system, this would also send a message to the tourist's app like "Your distress alert has been acknowledged, help is coming." We might simulate that by sending a push notification back to the app if time allows.

- **Multilingual and Accessibility:** By this phase, ensure the app supports at least one alternate language (for instance, Hindi). We'll translate the main UI text and allow switching. This demonstrates the feasibility of scaling to 10+ languages as required. Also, test the app's voice input or emergency hotkey (like the power-button triple press) for triggering SOS. On the dashboard side, we might not need multi-language (that's for officials who likely are fine with English, or could adapt to region). But for completeness, maybe support local language in sending messages to tourists.

By the end of Phase 2, the MVP should be **comprehensive and functional**: a tourist can register, get a digital ID, use the safety app with tracking and SOS, the system automatically detects certain problems, an authority can monitor on a dashboard, and an IoT device can tie in for added safety. This MVP can be showcased to stakeholders for feedback.

Phase 3: Pilot Deployment and Additional Features (Beyond MVP)
Once the MVP is validated, the next steps involve scaling and polishing into a real product:

- Migrate the digital ID system fully to a **production-grade blockchain network** (e.g. set up a Hyperledger Fabric consortium with nodes at central and state levels). Implement the **digital ID wallet** for tourists: possibly integrate with existing government ID like DigiLocker so tourists don't have to enter details already verified elsewhere. Also, implement **verification scanners** – for example, police at a checkpoint could have a mobile app to scan the tourist's QR code and verify authenticity against the blockchain record (Fabric can return a yes/no and details if authorized).

- Enhance **AI models** using data from pilot runs. For example, incorporate smartphone sensor data (accelerometer for falls, microphone for shouts) into anomaly detection. One idea is to use machine learning to predict the likelihood of an incident: e.g., given the sequence of a tourist's recent movements and contexts, output a risk level. We'd train that on historical incident data once available. Also implement **recommendation AI** – e.g., if a tourist's safety score is dropping, the app could proactively suggest "Avoid that route at night, it's less safe. Consider taking a taxi." This personalization could use AI models that learn from aggregate tourist behavior (this touches on travel personalization, which is outside core safety but could be value-add).

- Build out the **family/friends portal** so that a tourist can share a tracking link with someone. This could be a web view that shows the tourist's live location and status (only if they allowed it). It provides peace of mind to families. Could even integrate with messaging apps for automated periodic updates ("Your friend is at location X at 5 PM, all well").

- Implement robust **IoT device support**: possibly design custom wearable hardware (a rugged band with GPS, SOS, and GSM/LoRa connectivity for areas with no phone signal). This might involve partnership with a hardware vendor. Standardize the protocol so multiple devices can register and feed data.

- **Scalability & Security hardening:** Optimize the system for potentially thousands of tourists updating every few seconds. This might involve moving to cloud infrastructure, using load balancers, and separating services (e.g. a dedicated real-time location ingestion service). Also obtain necessary certifications for data security since this handles personal data. Perform penetration testing and ensure compliance with laws (e.g., explicit consent from tourists for tracking, option to opt out at any time which would disable tracking features, etc.). Possibly incorporate anonymization in analytics – e.g., when showing cluster heatmaps, maybe don't show personal identities unless zoomed in or an incident is flagged.

- **Integration with Emergency Services:** Work with local police to integrate with their control room systems. For example, connect our alert system to the emergency response system so that when an SOS comes, it automatically dispatches a police patrol or sends details to 112 centers with full info (currently 112 India does something similar with its "SHOUT" feature) [19]. Also, tie in local hospitals for medical emergencies – the app or dashboard could forward health data if a tourist needs rescue (like the heatstroke scenario, you'd alert nearest hospital with patient data) [20].

Each of these Phase 3 items would be its own project, but they give a sense of how the MVP can evolve into a production ecosystem.

For now, the **Phase 1 and 2 MVP deliverable** will suffice to prove the concept to the Ministry stakeholders: demonstrating real-time tourist monitoring, instant incident alerts, secure ID verification, and the feasibility of advanced tech like blockchain, AI, and IoT in one solution.

Product Requirements Sheet

To ensure clarity, here is a breakdown of the **key product requirements** for each component of the system. These encompass functional requirements (what the system should do) and some non-functional requirements (performance, security, etc.). This can serve as a checklist for development and evaluation:

A. Digital Tourist ID Platform (Blockchain-Based)

- **Tourist Registration**: The system shall allow authorized personnel (at entry points or online) to register a tourist by capturing personal identification (passport or Aadhaar number, etc.), a photo, emergency contact info, and trip details (itinerary, duration of stay) [2]. The tourist's consent for data use must be recorded.
- **Digital ID Issuance**: Upon registration, a unique digital ID will be generated for the tourist. This ID shall be stored on a blockchain ledger as a tamper-proof record, including at minimum: TouristID, validity period, and a hash or pointer to their personal details [2]. The tourist receives a QR code or digital token representing this ID (e.g., in their mobile app or via email).
- **Verification**: The platform must provide a means to verify a digital ID's authenticity and details by authorized users. For instance, scanning the QR code of a tourist should retrieve their basic info (name, photo) and verify it matches the blockchain record (e.g., by recomputing hash) [3]. This can be used by hotel check-ins or law enforcement in the field. Verification should work offline to some extent (e.g., using a stored public key to verify a signed credential) in case of connectivity issues.
- **Expiration/Revocation**: The digital ID must automatically expire after the approved trip duration. After expiration, it should no longer be valid for verification (verification attempts should indicate it's expired). There should also be a way for authorities to revoke an ID early (e.g., if a visa is cancelled or

a person is flagged) – revocation should update the blockchain record (perhaps via adding a "revoked" flag transaction).

- **Data Minimization**: The system should avoid storing sensitive personal data on-chain in plain form. Only minimal identity indicators or references are on blockchain [18], with full data stored securely off-chain (in an encrypted database). The link between on-chain and off-chain records must be secure (e.g., an ID number or DID).
- **Scalability**: The ID platform should be able to issue and manage a large number of IDs, e.g., thousands per day during peak tourist seasons, without significant delays. Blockchain transactions should commit within a few seconds if possible (permissioned networks can be tuned for faster finality). The design should consider throughput limits of the chosen DLT.
- **Audit Trail**: Every issuance, verification, or revocation event should be logged (the blockchain itself provides an audit trail for the ID data). Additionally, an admin log will record which official registered which tourist and any changes, for accountability.
- **Integration**: Provide APIs or SDK for the mobile app and dashboard to query ID status quickly. For instance, the app might show "Digital ID verified on blockchain" to the user with a green checkmark, which requires an API call.
- **Compliance**: The ID process must comply with KYC and privacy regulations. For example, if using Aadhaar, follow the UIDAI guidelines; ensure data storage follows the Personal Data Protection Act (with consent taken, data deletion after use, etc.). Also, since blockchain is immutable, ensure sensitive data isn't placed in immutable storage, or use encryption such that only authorized can decode it.

B. Tourist Mobile Application (Safety App)

- **User Authentication**: Tourists shall log in to the app using their digital ID or registered mobile number. The login process should be simple (possibly OTP-based or using credentials set at registration). This links the app instance to the tourist's identity securely.
- **Geo-location Tracking**: The app must gather the device's GPS location at regular intervals (configurable, e.g., every 5 minutes) and send it to the server when internet is available. It should also allow on-demand immediate updates (e.g., right after SOS). The tracking should run in the background even if the app is not open (with user permission).
- **Geo-fence Alerting**: The app will store or fetch a list of restricted or high-risk areas (defined by polygons or circles). It shall run geo-fencing services such that if the user enters one of these areas, an **on-entry alert** is triggered: the app shows a warning to the user and sends an alert to the server [6]. Ideally, even if the app is backgrounded, the OS should wake it on geo-fence trigger (Android has this capability). The app should also ideally notify when leaving such an area ("You have exited the restricted zone, please report to authorities if any issue occurred").
- **Panic SOS Button**: The app must have an always-accessible SOS feature. When activated (via in-app button tap, or hardware key shortcut, or voice command), it shall immediately do the following: **a)** send the current location and an SOS alert to the central server [21]; **b)** send SMS/email to the tourist's emergency contacts with a preset message and location link (if server supports SMS gateway); **c)** dial the local emergency number (112 in India) on speakerphone (this can be an optional setting, but likely useful to integrate) [8]. The UI should give feedback that an alert was sent ("Alert sent to police and family"). There should also be a way to cancel if it was accidental (e.g., within 5 seconds).
- **Real-Time Location Sharing (Opt-in)**: If the user enables live tracking (opt-in), the app should send location updates more frequently and allow designated people to view their live location. Possibly generate a shareable tracking link. The requirement is that when enabled, the location latency is low

(updates say every 10-15 seconds or streaming) and that it can be turned off anytime by the user for privacy.

- **Tourist Safety Score Display**: The app will display the current safety score (as a number and/or color/status) to the user [6]. If the score drops below certain thresholds, the app should also display context-aware tips (for example: "Your safety score is low because it's late night in a secluded area. We recommend staying in well-lit places or heading back to accommodation."). This encourages safer behavior. The app should fetch updated score from server (or calculate locally if rules are simple) whenever location updates or context changes.
- **Notifications and Messages**: The app should receive push notifications or in-app messages from the server. These could be warnings (e.g., "Heavy rain in your area, stay cautious") or direct messages from authorities (maybe in response to an SOS: "Police en route, ETA 10 mins"). Thus, the requirement is to integrate a push messaging system (could be Firebase Cloud Messaging for ease) for urgent communications. Multi-language support is important here too – messages should ideally come in the user's preferred language.
- **Multilingual Support & Accessibility**: All user-facing text in the app should be translatable. The MVP will support English and one or two Indian languages, but the architecture should accommodate 10+ languages easily (via resource files or localization frameworks). The app should also accommodate different literacy levels – using icons and simple UI for critical functions (e.g., an SOS icon button that's universally understandable). Voice assistance (like speaking the alert "Entering restricted zone!") in the local language would be a plus for accessibility. For differently-abled travelers, the app should be usable with screen readers (for visually impaired) – meaning UI elements have proper labels.
- **Offline Behavior**: If the user is offline (no internet), the app should still function to the extent possible. For instance, geo-fence alerts can still warn the user locally even if it can't notify the server immediately. The app should queue any SOS or location updates and send them once connectivity returns (with a timestamp). Perhaps integrate SMS fallback: if completely offline but a cell network is present, the app could try to send an SOS via SMS to a predefined number with GPS coords. This might require a separate arrangement, but is worth considering for true emergency resilience.
- **Privacy Controls**: The app must clearly allow the user to pause tracking or adjust what is shared. For example, a user might disable family sharing even if they keep authorities sharing on. Also a "Delete my data" option after trip (which would remove their info from the app and send a request to server for deletion of off-chain records, while blockchain might retain an anonymized proof). Ensuring users feel in control of their data will improve adoption.

C. AI Anomaly Detection Module

- **Location Monitoring**: The system shall continuously collect and store the sequence of location updates for each tourist (with timestamps).
- **Rule-Based Alerts**: Implement baseline rules that generate alerts: no check-in for X hours, entering known dangerous area, deviating Y km from itinerary, sudden stop after high-speed travel, etc. [10]. These thresholds (X, Y) should be configurable by admins and possibly personalized (e.g., an elderly tourist might trigger inactivity alert sooner than a younger backpacker).
- **Machine Learning Capabilities**: The system should be designed to accommodate ML algorithms on the data. For MVP, maybe not fully in use, but requirements for future: The system should be able to mark data as "normal" or "incident" to build a training set. It should support plugging in an ML model that can analyze a tourist's pattern (perhaps a daily path or speed profile) and output an anomaly score. If above a certain threshold, an alert is triggered. In the future, multiple data sources

(wearable vitals, local weather/disaster data, social media alerts for area, etc.) could be fused – the architecture should allow adding these inputs.

- **False Positive Minimization**: It's important the anomaly detection doesn't overwhelm authorities with false alarms. The requirement is to implement a **validation step** or escalation logic. For example, if an anomaly is detected, the system could first send a push notification to the tourist: "We noticed you haven't moved in a while, are you okay?" If the tourist responds "Yes, I'm fine", the system can abort the alert (or mark it low priority). Only if no response (or a negative response) does it escalate to police [10]. This two-step confirmation can be built in to improve accuracy.
- **Automated E-FIR**: If a tourist is deemed missing (e.g., no contact > 24 hours, and attempts to reach them fail), the system should automatically compile a **Missing Person report**. This includes their identifying details, last known location/time, and any contextual info (last seen with someone? vehicle info if available, etc.) [12]. While actual FIR filing might need human action, the system can at least generate a document that officers can quickly use to file officially. The requirement is the report should be comprehensive and in a format acceptable to police (maybe a PDF form).
- **Learn & Adapt**: The AI module should improve over time. So, it should store outcomes – e.g., after an alert, if it turned out to be a real emergency or a false alarm – so that future models can learn. This implies maintaining a dataset of incidents and non-incidents. Possibly integrate with feedback: allow authorities to label an alert as false alarm or genuine. This data is then fed into refining the rules or model thresholds.

D. Tourism Department & Police Dashboard

- **User Management (Admin)**: The dashboard shall have secure login for different roles: e.g. Police user, Tourism officer, System admin. Police users can view and manage alerts, view tourist info; Tourism officers can view stats, cluster maps, but maybe not personal emergency data; Admin can manage system parameters (geofences, user accounts, etc.).
- **Real-Time Map View**: The dashboard must provide a map with real-time (or near real-time) positions of tourists. It should update automatically as new data comes (without full page reload). The map should allow zoom/pan and have layers to toggle (satellite view, heatmap). It should clearly mark high-risk zones on the map (perhaps shaded red areas) so officials see if tourists are inside them [22]. There should be a legend and maybe filter options (like highlight only tourists with alerts or with low safety score).
- **Tourist Detail View**: When an official selects a tourist (via search or clicking a map marker), the dashboard shall display that tourist's profile: photo, name, age, nationality, digital ID number, emergency contact, itinerary (perhaps in a timeline or list of destinations), current safety score, and any past alerts involving them. This helps in quickly briefing responders. Sensitive info like passport number can be hidden or shown on demand to avoid on-screen exposure unless needed.
- **Alerts Management**: The dashboard shall prominently show active alerts/incidents. Each alert entry will have: timestamp, type (SOS, GeoFence, Inactivity, etc.), tourist identity, location of alert, and status (new/acknowledged/resolved). Officials can click an alert to see more info and then mark it **Acknowledged/Under Response**, adding a note (e.g., "Dispatched team to location"). The system should log who acknowledged it and when. Once resolved, it can be closed with a resolution note ("Tourist found safe"). Resolved alerts move to history logs. This workflow ensures accountability and avoids duplication (multiple officers responding separately unaware).
- **Analytics & Visualization**: The tourism department users would want aggregated data: e.g., total tourists currently in state, number of alerts in last 24h, distribution of tourists by region, popular spots vs. risky spots, etc. The dashboard should have a section for **analytics** – charts or reports. For MVP, this could be basic (like a counter of active tourists, count of SOS alerts so far). In a full product,

this can evolve to daily/weekly reports, trend lines (e.g., “Incidents per 100 tourists” metric), etc. If needed, exportable data (CSV) for further analysis by departments.

- **Heat Maps & Clusters**: Visualizing clusters of tourists can help identify if crowds are building up (which might require crowd control or could be targets for criminals). The system should be capable of showing a **heat map** of tourist density – maybe using intensity gradients on the map. Also, a heat map of **high-risk zones** highlighting where incidents frequently occur or areas flagged by AI as needing attention. This can guide preventive measures.
- **Search and Filter**: A search bar to lookup a specific tourist by name or ID is required (for example, if an embassy calls asking about a particular citizen, officials can quickly find if they are in the system and their status). Also filters on the map/list: by nationality (if needed), by travel company (if they came in a group tour), by date of entry, etc., which could aid in targeted operations (like sending advisory to all tourists from a certain country in their language).
- **System Dashboard Performance**: It should refresh data reasonably quickly (map positions updated, say, every 30s or faster when zoomed in). However, it must also be efficient – thousands of points might slow browsers, so using clustering or only showing details on demand is needed. The system should be tested for, say, 5,000 concurrent tourist points and 50 concurrent alerts without crashing or lagging significantly. If needed, paging in list views (show 100 at a time etc.).
- **Security & Access Control**: The dashboard must only be accessible over secure connections (HTTPS, perhaps VPN if internal). Different roles see different data – e.g., a tourism dept user might not see personal IDs, whereas police do. Also, every view or action should be logged in an admin audit log (for example, “User X viewed details of Tourist Y at time Z”). This is important when dealing with sensitive identity info. We should also implement session management (auto log-out after inactivity) to reduce risk of unattended terminals.
- **Emergency Coordination Features**: A nice addition would be an integrated chat or comment system on each incident, where multiple officers can coordinate (like one updates “I’m at the site with the tourist”). This could be considered but for MVP maybe not. At minimum, provide contact info of tourist’s emergency contact in the incident view so police can click to call family if needed. Possibly integrate an API to directly message the tourist from the dashboard (“send instruction to tourist’s phone”).

E. IoT Wearables Integration

- **Device Registration**: The system shall allow linking an IoT device to a tourist’s profile (likely at the time of tourist ID issuance). For example, scanning a device QR code or inputting its unique ID (MAC address or serial) into the system to pair it with the tourist. A tourist could have 0, 1, or multiple devices (maybe a band and a smart backpack, etc., though initially 1 is enough).
- **SOS Signal**: The wearable device must be able to send an SOS alert that the system receives within seconds. Whether through the phone or direct internet, the requirement is that pressing the button leads to an alert just like the app’s panic button. The system should recognize the alert’s source (e.g., “Device SOS from Tourist 1234”) and process it equivalently [14].
- **Health/Telemetry Data**: The device should periodically send data like location (if it has GPS or can approximate via phone Bluetooth), heart rate, or movement status. The system should ingest this and attach it to the tourist’s record. For MVP with ESP32, we’ll simulate perhaps heart rate. The requirement is that the system can handle continuous data streams from devices without losing information. If a dangerous threshold is crossed (like heart rate too high/low, or a fall detected by accelerometer), the device/client code should send an immediate alert which the backend interprets (e.g., an alert of type “Health Alarm”). For instance, if a band detects no pulse (hopefully just hypothetical), it could send an “SOS-Health” alert prompting medical rescue.

- **Low Connectivity Operation**: The design should consider that IoT devices might connect via a tourist's smartphone (BLE) or via independent network. If via phone, ensure the app can relay device data even in background. If independent (like a LoRaWAN gateway at a forest ranger station), ensure integration with our server (maybe via an IoT cloud service or direct internet if device has SIM/WiFi). MVP will assume connectivity through phone Wi-Fi hotspot. The device should also have a keep-alive; if the server hasn't heard from a device in a while, it might alert that the device is offline (which could mean it's out of range or broken).
- **Battery and Maintenance**: Though not software per se, the product requirements should note that wearables need to last long enough (e.g., 1 week battery) and be weather-proof if used in outdoors. For our ESP32 prototype, we'll keep it powered via USB, but a real product might use low-power modes. The system can also alert if a device's battery is low (if the device sends battery level).
- **Ease of Use**: The wearable should be simple – one button for SOS, maybe one light to indicate status. It's meant for possibly elderly or non-tech-savvy tourists as well, so it must be foolproof (no complex operations). In software terms, that means designing the pairing to be straightforward (like scanning a QR and it auto-registers) and ensuring false alarms are minimized (debounce the SOS button, maybe require 2-second press to activate to avoid accidental presses).
- **Optional Integration**: The MVP demonstrates IoT but if in production not every tourist will have this. It might be optional or given in certain areas. The system should be designed modularly so that it works with or without IoT input. If a tourist has no device, the platform should still cover them via the phone. If they do have a device, it's supplementary data.

F. Non-Functional Requirements (Cross-cutting)

- **Safety and Reliability**: The system is mission-critical (people's lives could depend on it), so it must be reliable. Uptime target should be high (e.g., 99.9%). There should be redundancies – e.g., if one server node fails, another picks up (cloud deployment in multiple regions possibly). Similarly, data should be backed up (especially the digital ID records and logs). We cannot afford data loss, especially if an incident is ongoing.
- **Performance**: Real-time responsiveness is key. SOS alerts should propagate from app to dashboard ideally within **<5 seconds** end-to-end. Geofence alerts similarly quick. The UI updates should feel instantaneous. The system should handle possibly hundreds of concurrent alerts in worst-case (like a disaster scenario affecting many tourists at once) – though in normal operation alerts are sparse. Scaling out the server and using efficient messaging (WebSockets, etc.) will help meet this.
- **Security**: The entire system must be secure against cyber threats. All APIs authenticated (e.g., app uses token authentication to send data, dashboard uses user accounts). Protect against spoofing (e.g., someone should not be able to send a fake SOS by imitating device or app without the right credentials). Use encryption everywhere (SSL/TLS). Regular security audits should be part of maintenance. Given the involvement of personal data and government, compliance with standards like ISO 27001 could be aimed for eventually.
- **Privacy**: As mentioned, strict data privacy measures are needed. Only minimum necessary data is shared with authorities (for instance, police might not see full itinerary unless needed, etc.). Tourists should be informed what data is collected and for what purpose (likely via an in-app consent form on first use). The design should avoid any function creep (like using this data for tourism marketing without consent, etc.). It's purely for safety, and that principle should reflect in data handling.
- **Maintainability & Extensibility**: The platform should be built with modular architecture (microservices or at least logical separation) so that new features like additional IoT sensor types, or integration with new data sources (e.g., weather alerts API to cross-check environmental dangers)

can be added without a complete overhaul. Also, code should be well-documented and follow best practices, as it might be handed over to NIC or a government IT team to maintain long-term.

- **User Experience**: For tourists, the app should be as **unobtrusive as possible**. It should not drain battery significantly – which means optimizing GPS usage and using low-power geofence methods. It should also work on mid-range devices that many tourists might have. The UI should be intuitive: even if someone doesn't go through a tutorial, the SOS and warnings should be self-explanatory. For the authorities, the dashboard UX should prioritize clarity – showing what needs attention first (like active incidents blinking at top). Training for operators should be minimal – a quick guide should suffice to use the interface.
- **Field Testing**: The system should be tested in real-world pilot (maybe one Northeast state) before full rollout. Requirements from field include working in low-connectivity scenarios. Possibly incorporate SMS or radio as backup channels. The IoT wearable should be tested on a trek to see if it stays connected. We likely need to fine-tune based on pilot feedback (e.g., adjust geofence sensitivity, adjust anomaly thresholds to local conditions, etc.).

By adhering to these requirements, we aim to create a **robust, user-centered MVP** that can later be built out into a full-scale Smart Tourist Safety Monitoring & Incident Response System. The approach combines **proven technologies** (mobile apps, GPS tracking, digital IDs) with **innovative integrations** (AI for anomaly detection, Blockchain for identity authenticity, IoT for on-field safety) to address the problem statement in a comprehensive manner [23] [24]. The end result will be a safer environment for tourists – they can travel with confidence knowing help is one tap away – and a powerful tool for authorities to ensure tourism remains safe and well-managed, which ultimately benefits everyone.

References & Inspiration Sources

- The problem statement itself guided much of this design, outlining features like geo-fencing alerts, AI anomaly detection, and blockchain IDs [25] [24]. Our implementation plan expands on these with practical tech choices.
- **Digital ID & Blockchain**: The concept of a blockchain-based traveler ID is inspired by initiatives like the **Known Traveller Digital Identity (KTDI)** pilot backed by Canada/Netherlands, where travelers carry verifiable digital credentials for faster, secure processing [3] [4]. Blockchain provides a decentralized, tamper-proof way to manage identities, which we leverage for trust and security of tourist IDs [5]. A reference in SecureIDNews highlights how blockchain can help authenticate passengers by sharing data via an app with authorities while keeping data secure [3]. This aligns with our digital ID platform for tourists.
- **Mobile Safety Apps**: Existing emergency apps like **112 India** (India's all-in-one emergency app) demonstrate the effectiveness of quick SOS alerts with location [8]. Our panic button draws from such real-world systems. Similarly, many women's safety apps (e.g., HollieGuard in UK, Noonlight in the US) influenced features like live location sharing and automatic alarm triggers. These gave us confidence that implementing an SOS with live tracking is feasible and socially accepted.
- **AI and Anomaly Detection**: In the travel domain, AI is increasingly used for security and personalization. For example, anomaly detection is used to flag unusual patterns in various sectors. A DjangoStars article on AI in travel mentions **"Enhanced Security – AI/ML in travel powers anomaly detection and behavioral authentication"**, which supports our idea of using machine learning to detect unusual tourist behavior for safety [11]. While that context was more about fraud, the principle is the same – ML can identify out-of-the-ordinary patterns that warrant investigation. Academic research like the Appl. Sciences 2023 paper by Ngeoywijit et al. (T3S system for tourist

safety in hot climates) shows integrating health data (e.g., smartwatch detecting heatstroke risk) and rapid alerts to medical services can save lives [20]. This influenced our inclusion of wearable health monitoring and connecting to rescue services.

- **IoT Wearables**: The use of wearables for safety is inspired by existing devices and trends. Many smartwatches have fall detection that auto-calls help – Apple Watch's feature is a prime example. In our research, a blog on IoT in wearables notes **"devices come equipped with GPS tracking and emergency response features... smartwatches have fall detection that alerts contacts if the wearer can't get up"** [15], reinforcing that our IoT band concept is quite realistic. We also looked at solutions like personal safety pendants and kids' trackers for design cues (simple one-button devices).
- **Government Initiatives**: We noted that Meghalaya (a NE state) launched a **Digital Guest Monitoring system** for tourists where accommodations log visitor details online for security [26] [27]. This indicates government interest in digitizing tourist tracking, though our system is more advanced (real-time tracking vs. lodging records). It shows that our project aligns with ongoing efforts to boost tourist safety through technology.
- **Tech Stack Choices**: We considered standard industry practices for large systems. For example, **Hyperledger Fabric** is used in several government blockchain projects due to its permissioned nature and privacy controls, which suits our case [5]. Flutter's popularity for cross-platform apps with good performance influenced our mobile strategy. Node.js and Python are both widely used in hackathons and enterprise for quick development and AI tasks respectively – an approach of combining them is common in modern architectures.
- **Data Protection and Ethics**: We also drew from data privacy resources to ensure compliance – e.g., the UK's ICO guidance on DLT and data protection emphasizes careful handling of personal data on blockchain [28]. We thus decided on minimal on-chain data and user consent mechanisms.

By synthesizing ideas from these sources and real-world examples, we ground our solution in proven technology and practices while pushing innovation. The result is a strategy that is ambitious yet feasible, and an MVP plan that is scoped to demonstrate the most important features effectively. With this roadmap, we can confidently proceed to development, creating a product that could significantly enhance tourist safety and incident response in the targeted regions and beyond.

Sources Cited:

- Smart Tourist Safety Monitoring System – Problem Statement (SIH 2025) [25] [24]
- SecureIDNews – *Blockchain-enabled ID for travel (KTDI pilot)* [3] [4]
- 112 India App – *Emergency SOS alert features* [8]
- DjangoStars Blog – *AI/ML Transforming Travel (security/anomaly detection)* [11]
- Zazz IoT Wearables Blog – *Wearables for personal safety (GPS & fall detection)* [15]
- Meghalaya Tourism Dept – *Digital Guest Monitoring initiative* [26] [27]

---

[1] [26] [27] Meghalaya Launches Digital Guest Monitoring System for Tourism Security – Pragyanxetu since 2013: APSC Study Materials Hub
https://pragyanxetu.com/meghalaya-launches-digital-guest-monitoring-system-for-tourism-security/

[2] [6] [7] [10] [12] [14] [21] [22] [23] [24] [25] Copy of SIH2025 Complete Data | PDF | Pesticide | Agriculture
https://www.scribd.com/document/909018841/Copy-of-SIH2025-Complete-Data

[3] [4] [13] Can blockchain-enabled ID make travel more secure and efficient? - SecureIDNews
https://www.secureidnews.com/news-item/can-blockchain-enabled-id-make-travel-secure-efficient/

[5] [17] [18] Blockchain-Enabled Digital Identity: Secure & User-Centric
https://www.rapidinnovation.io/post/blockchain-enabled-digital-identity-solutions-revolutionizing-privacy-and-security-in-2024

[8] 112 India on the App Store
https://apps.apple.com/us/app/112-india/id1441951304

[9] Emergency Response Support System in India - 112.gov.in
https://112.gov.in/

[11] AI/ML in travel: smarter, faster, data-driven journeys ✈
https://djangostars.com/blog/machine-learning-in-travel-industry/

[15] [16] IoT in Wearables: Benefits, Applications, and Latest Trends
https://www.zazz.io/blog/iot-in-wearables/

[19] 112 India Mobile App | How to use panic button - YouTube
https://www.youtube.com/watch?v=zCZfvffsTTE

[20] A Mobile Solution for Enhancing Tourist Safety in Warm and Humid Destinations
https://www.mdpi.com/2076-3417/13/15/9027

[28] [PDF] Blockchain applications in the United Nations system: towards a ...
https://docs.un.org/en/JIU/REP/2020/7